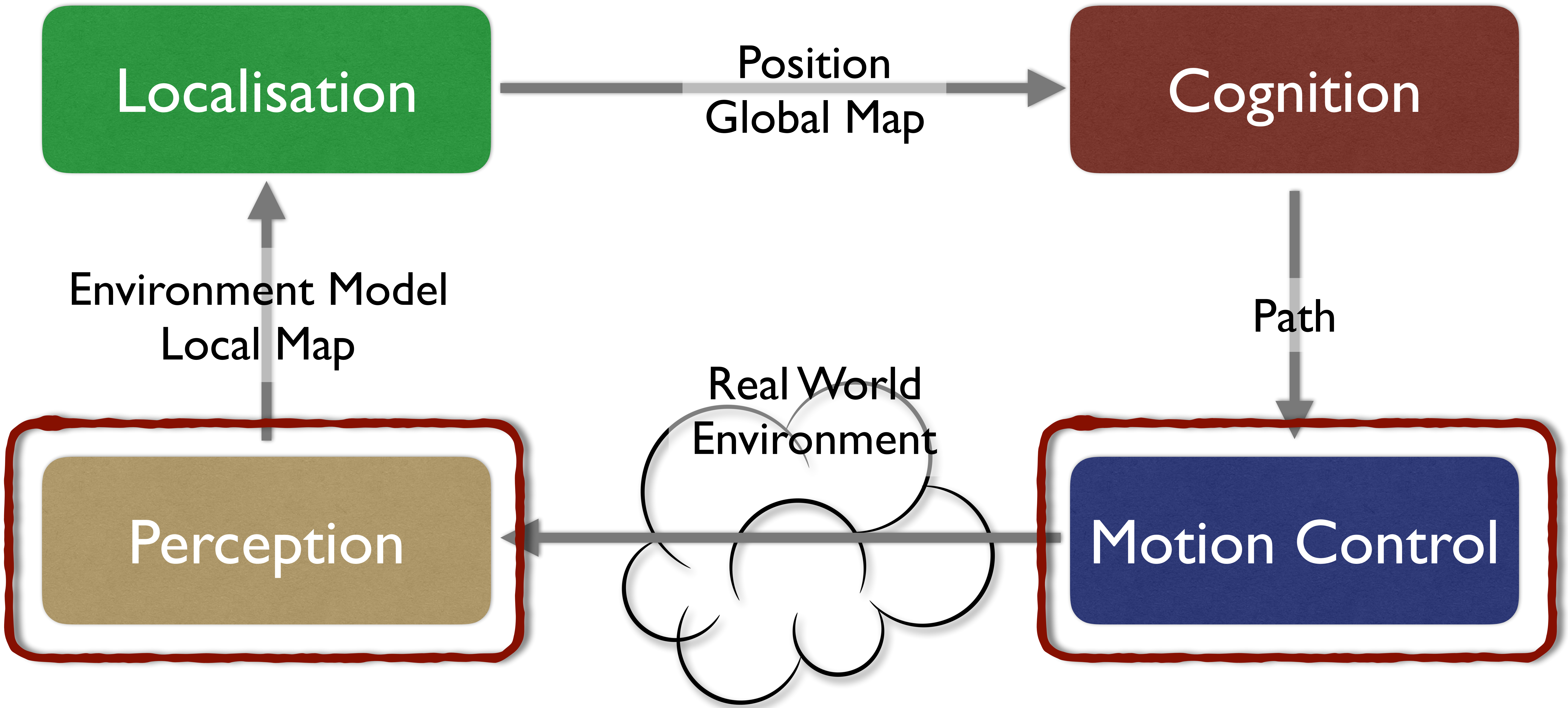
COMP329 Robotics and Autonomous Systems

Lecture 5: Perception and Odometry

Dr Terry R. Payne
Department of Computer Science
University of Liverpool

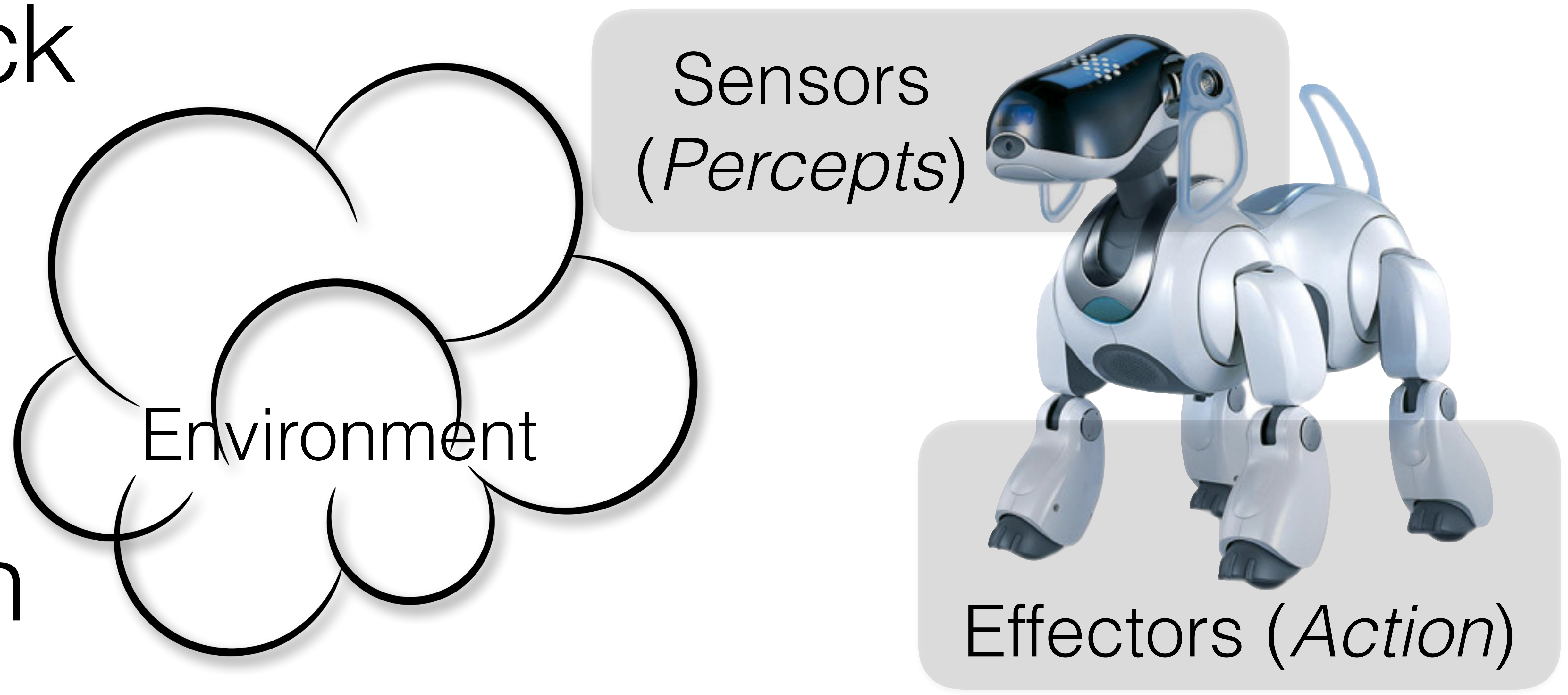
General control architecture



Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Perception

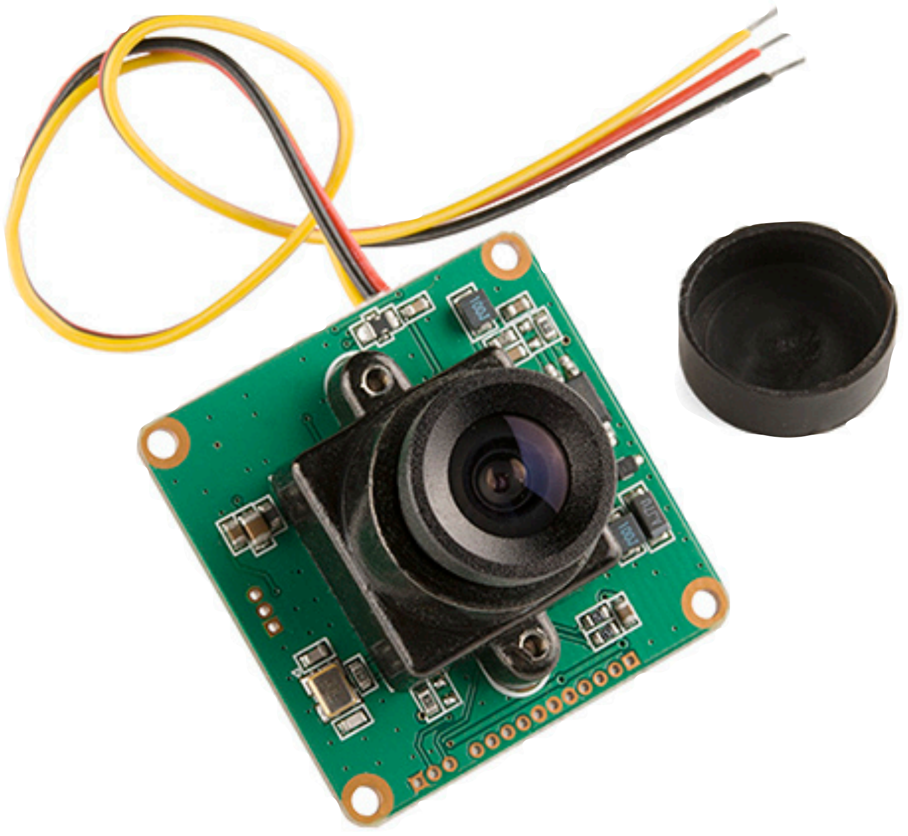
- Sensors give important feedback from the environment
  - Without them, robots are blind
- Perception is all about what can be sensed and what we can do with that sensing



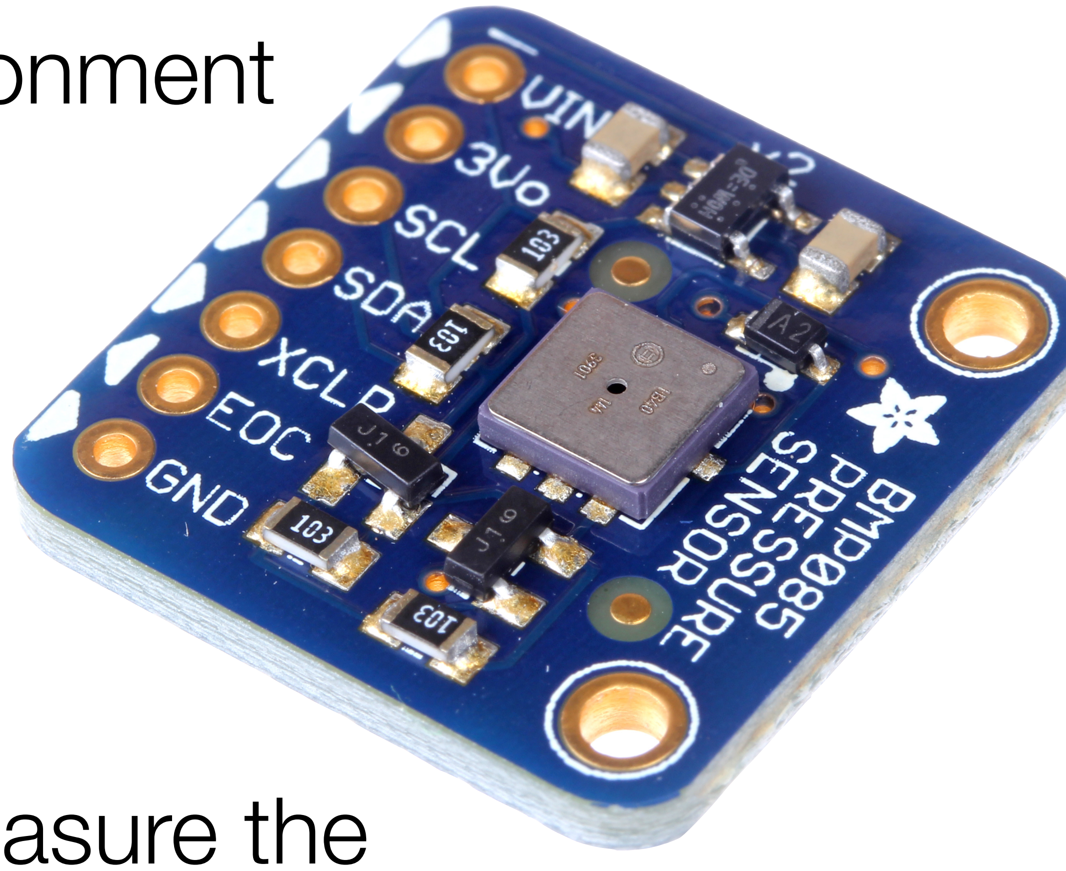
Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Classification of Sensors

- Proprioceptive sensors
  - Measure values internally to the system (robot),
    - (motor speed, wheel load, heading of the robot, battery status)
- Exteroceptive sensors
  - Information from the robots environment
    - (distances to objects, intensity of the ambient light, unique features.)



- Passive sensors
  - Energy coming from the environment



- Active sensors
  - Emit their own energy and measure the reaction
  - Better performance, but some influence on environment

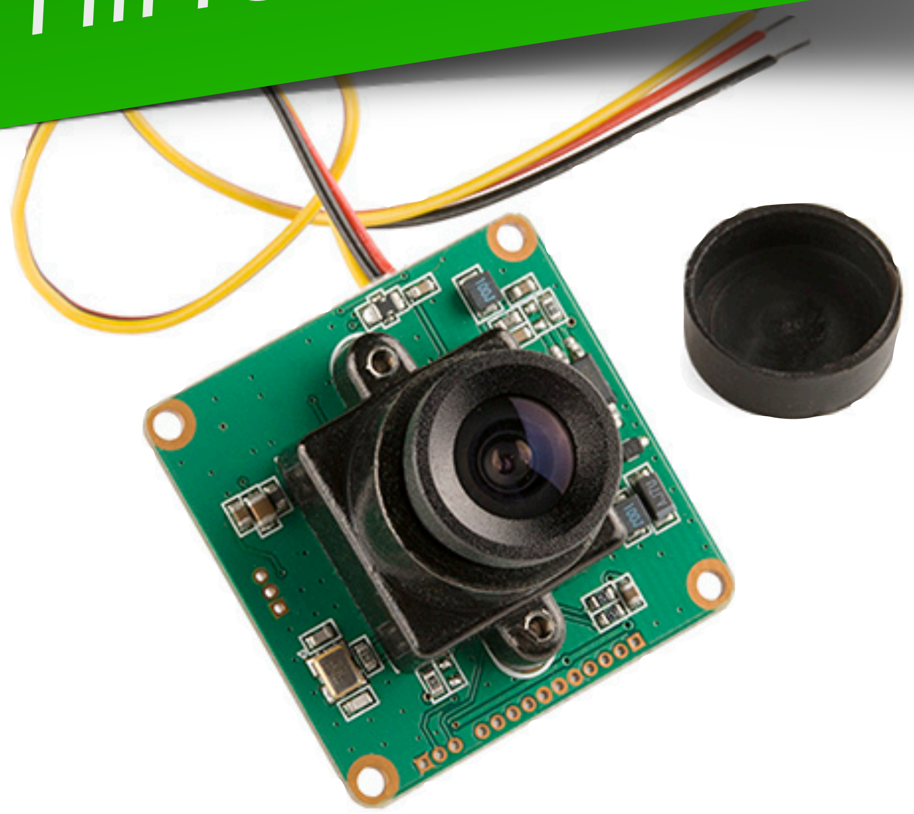
Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Classification of Sensors

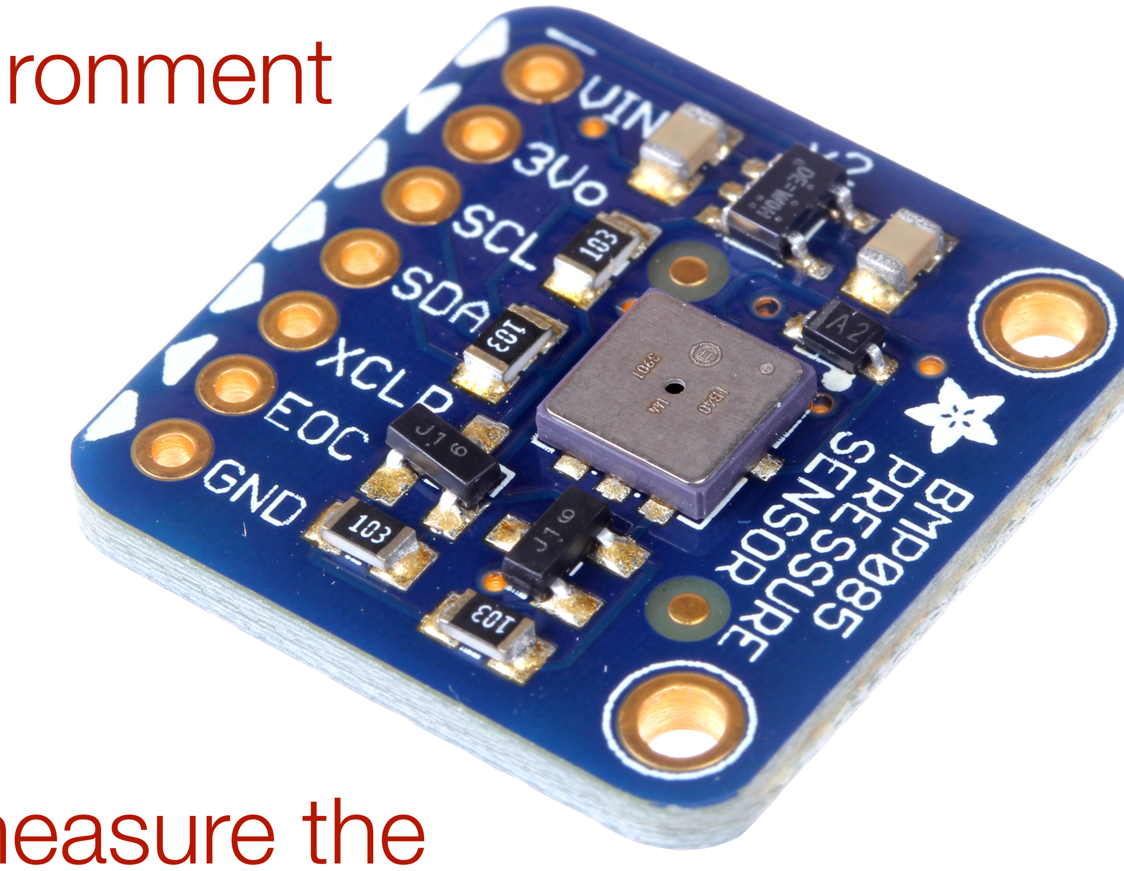
- Proprioceptive sensors
  - Measure values internally to the system (robot),
    - (motor speed, wheel load, heading of the robot, battery status)

- Exteroceptive sens

Focus of today's lecture; sensors that the robot uses to determine its state.



- Passive sensors
  - Energy coming from the environment



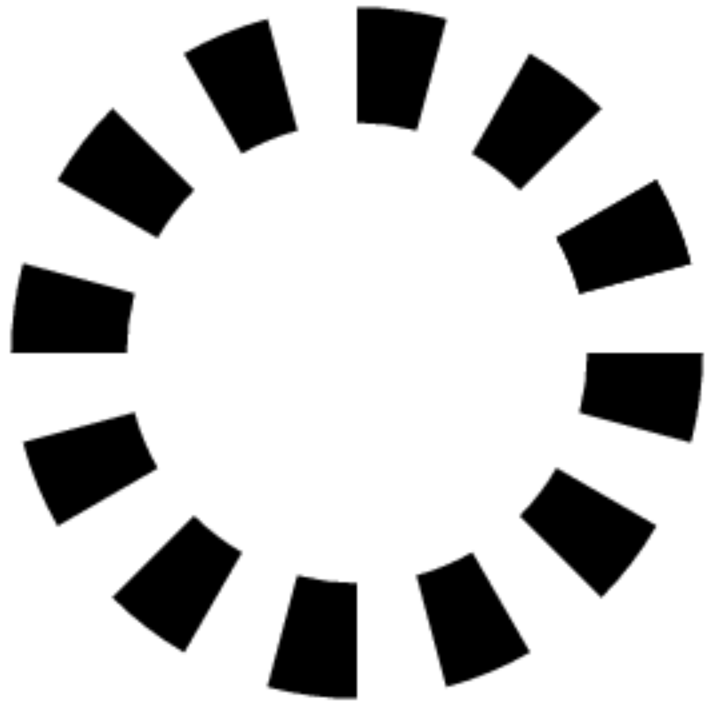
- Active sensors
  - Emit their own energy and measure the reaction
  - Better performance, but some influence on environment

Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

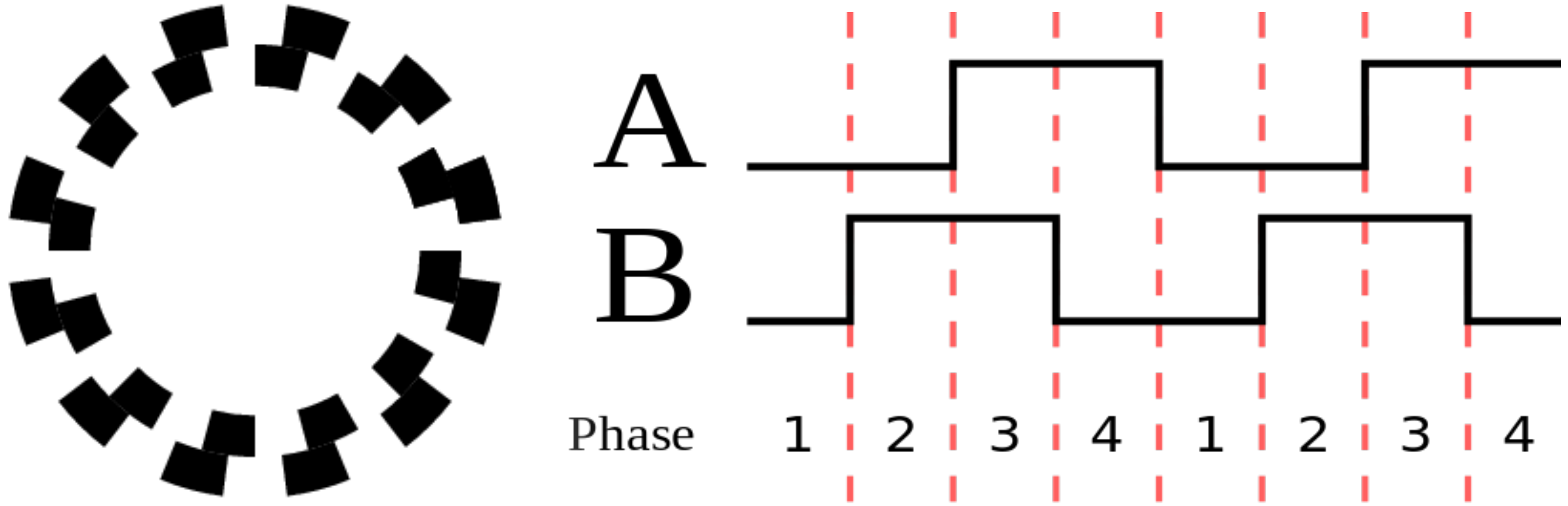
Wheel Encoders

- Measure position or speed of the wheels or steering.
  - Wheel movements can be integrated to get an estimate of the robot's position
    - ***Odometry***
- Optical encoders are ***proprioceptive*** sensors
  - Position estimate is only useful for short movements.
  - Typical resolutions: 2000 increments per revolution.
- Count the changes from black to white:
  - Measure light passing through the encoder.
  - Bounce light off the encoder.

- Simple encoder will give you count/speed.

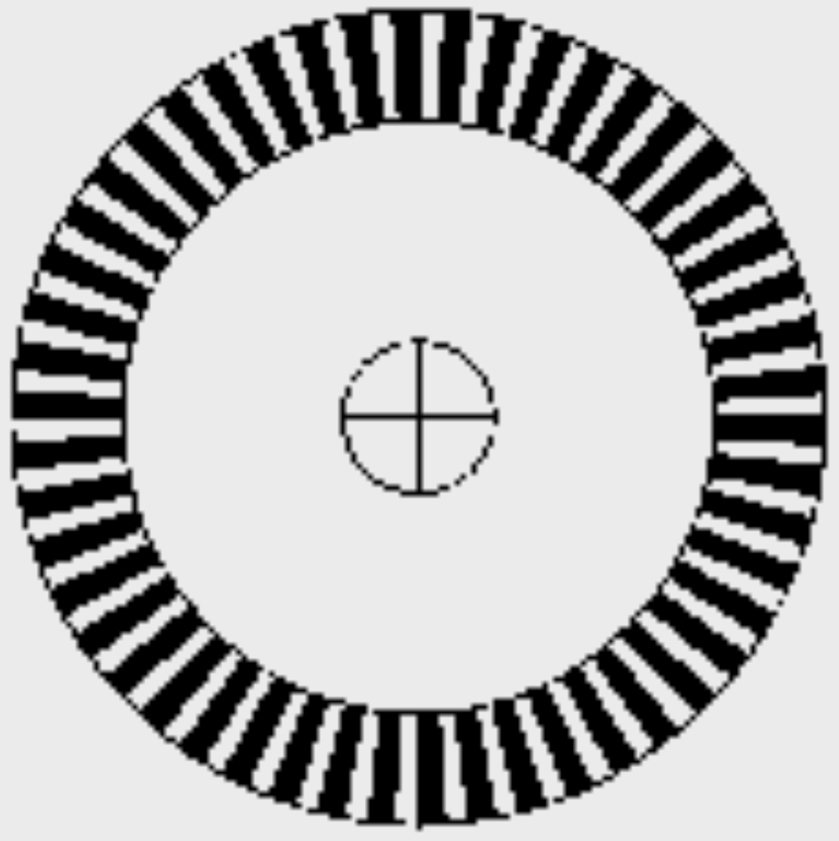


- Quadrature encoder will you direction also.
  - Look at phase of signals from the two bands on the encoder.

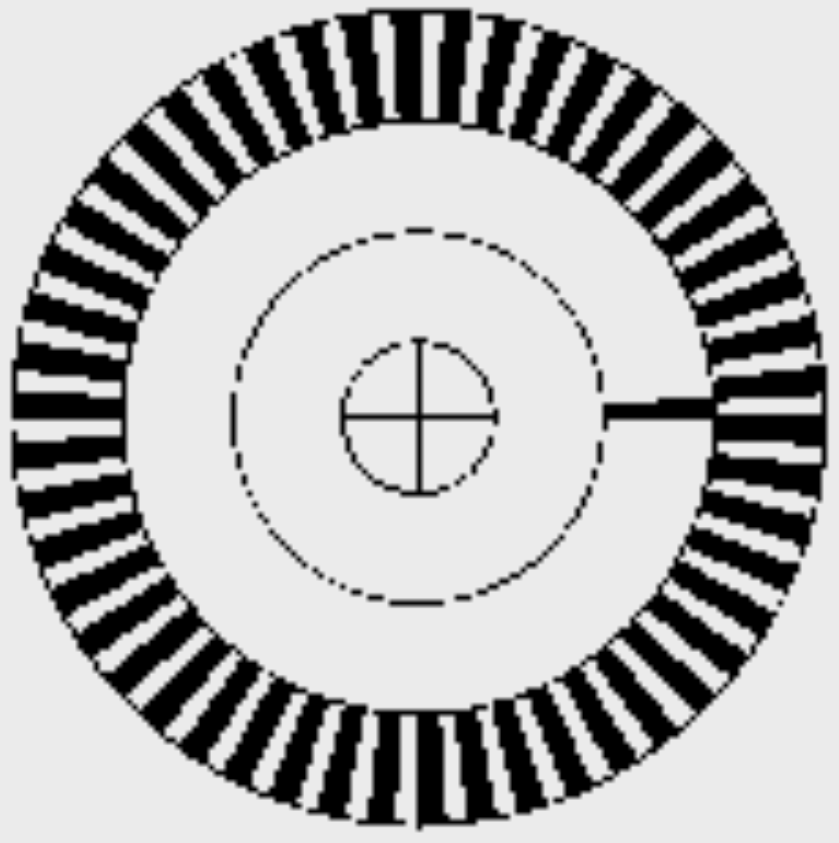


Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

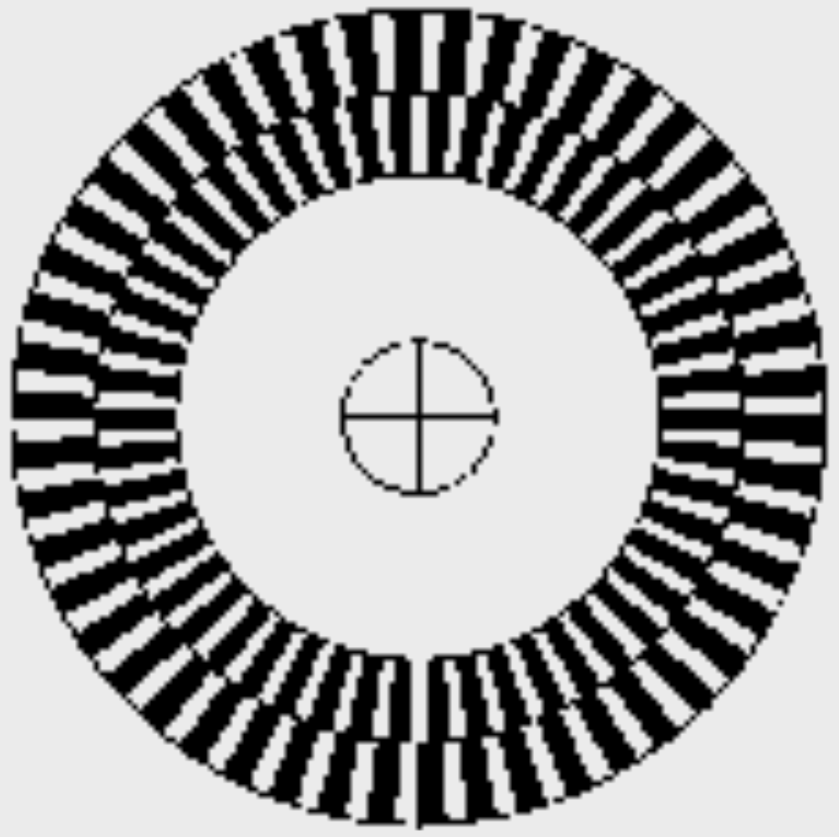
Wheel Encoders



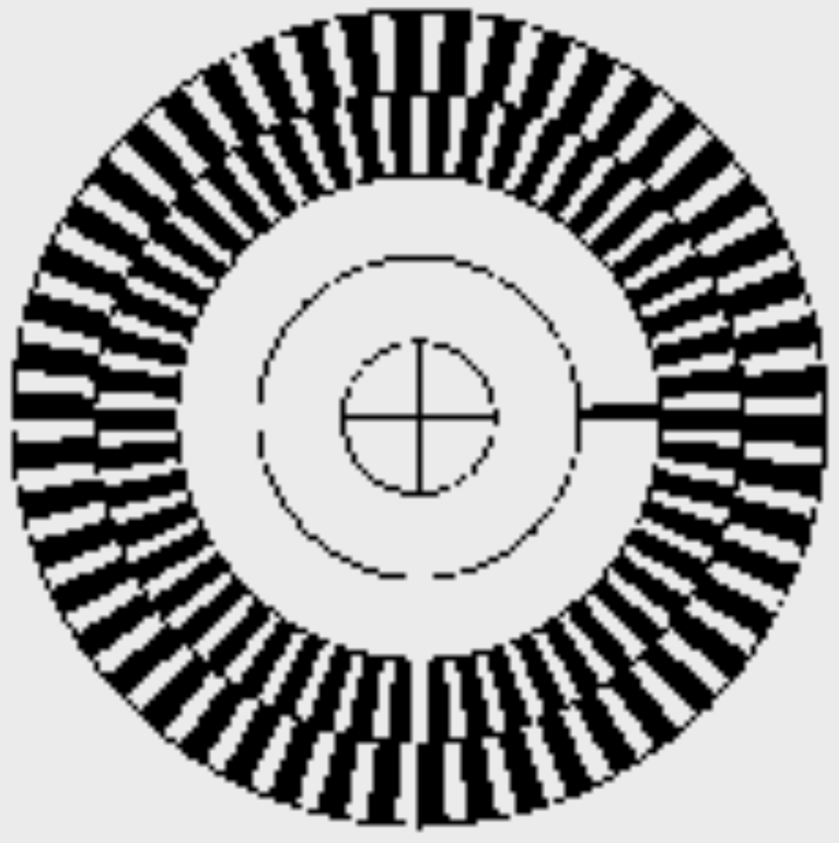
Unencoded



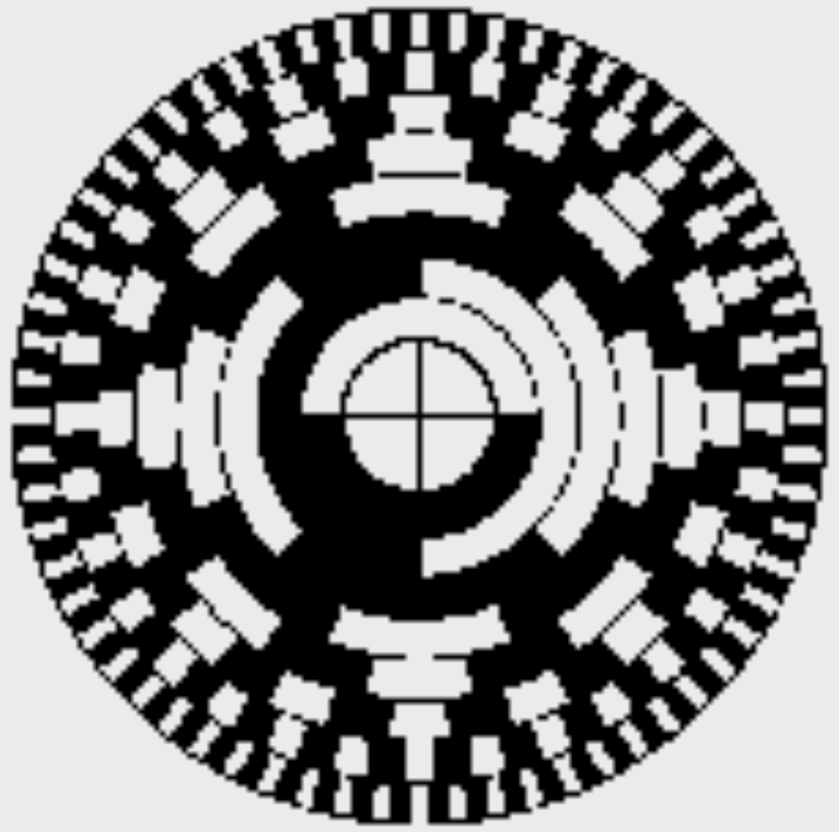
Unencoded with Index Track



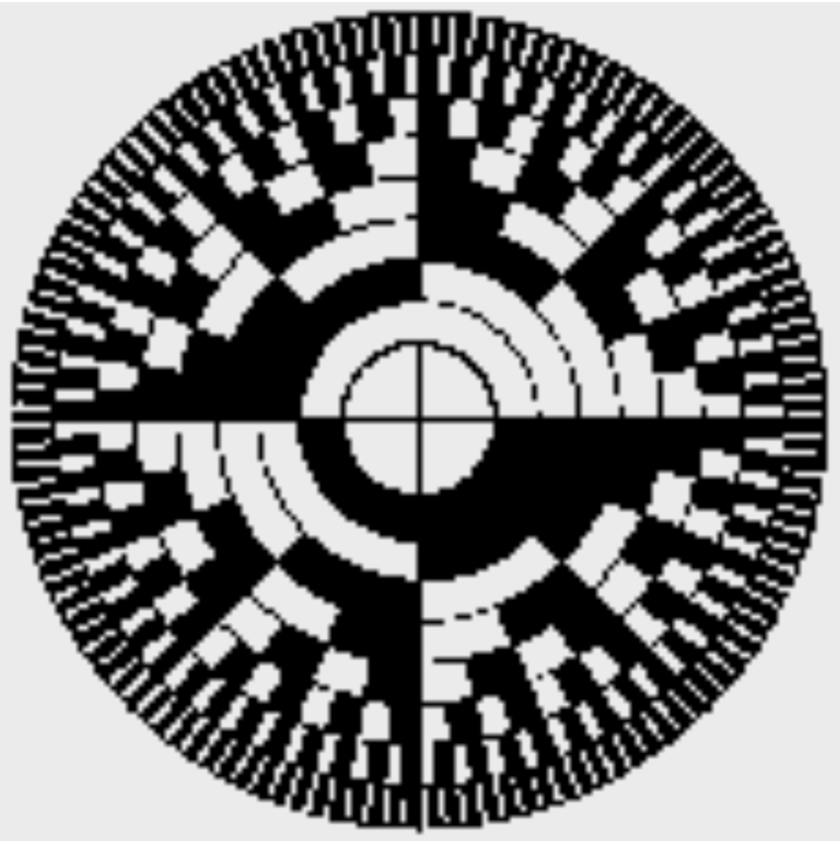
Two Track Quadrature



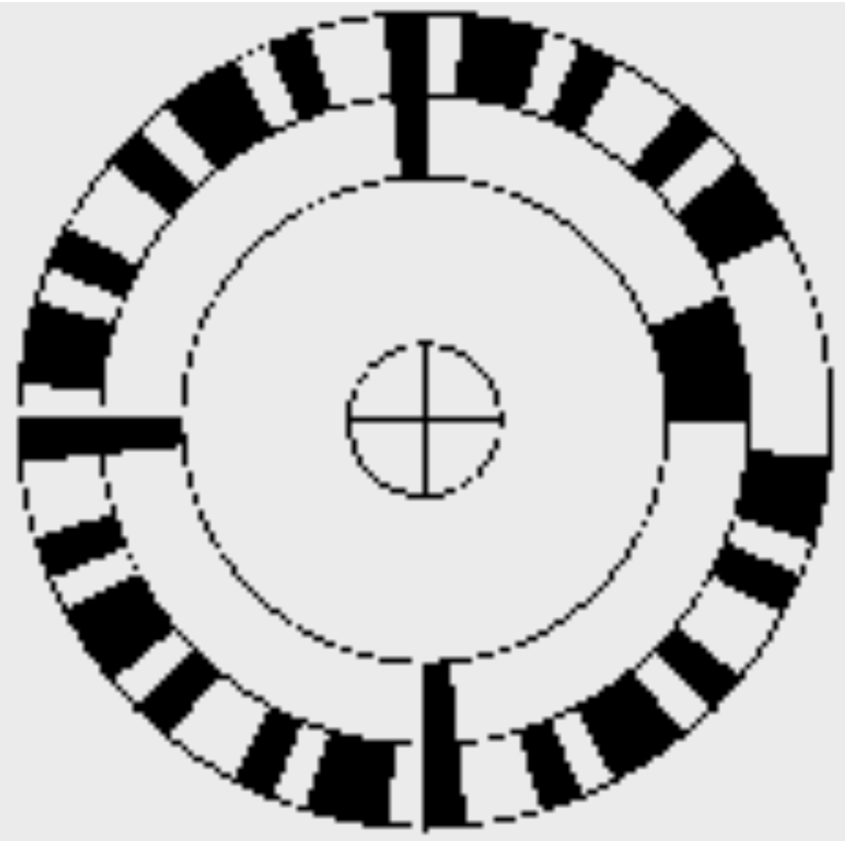
Two Track Quad with Index Track



Absolute Position Gray Code



Absolute Position Binary

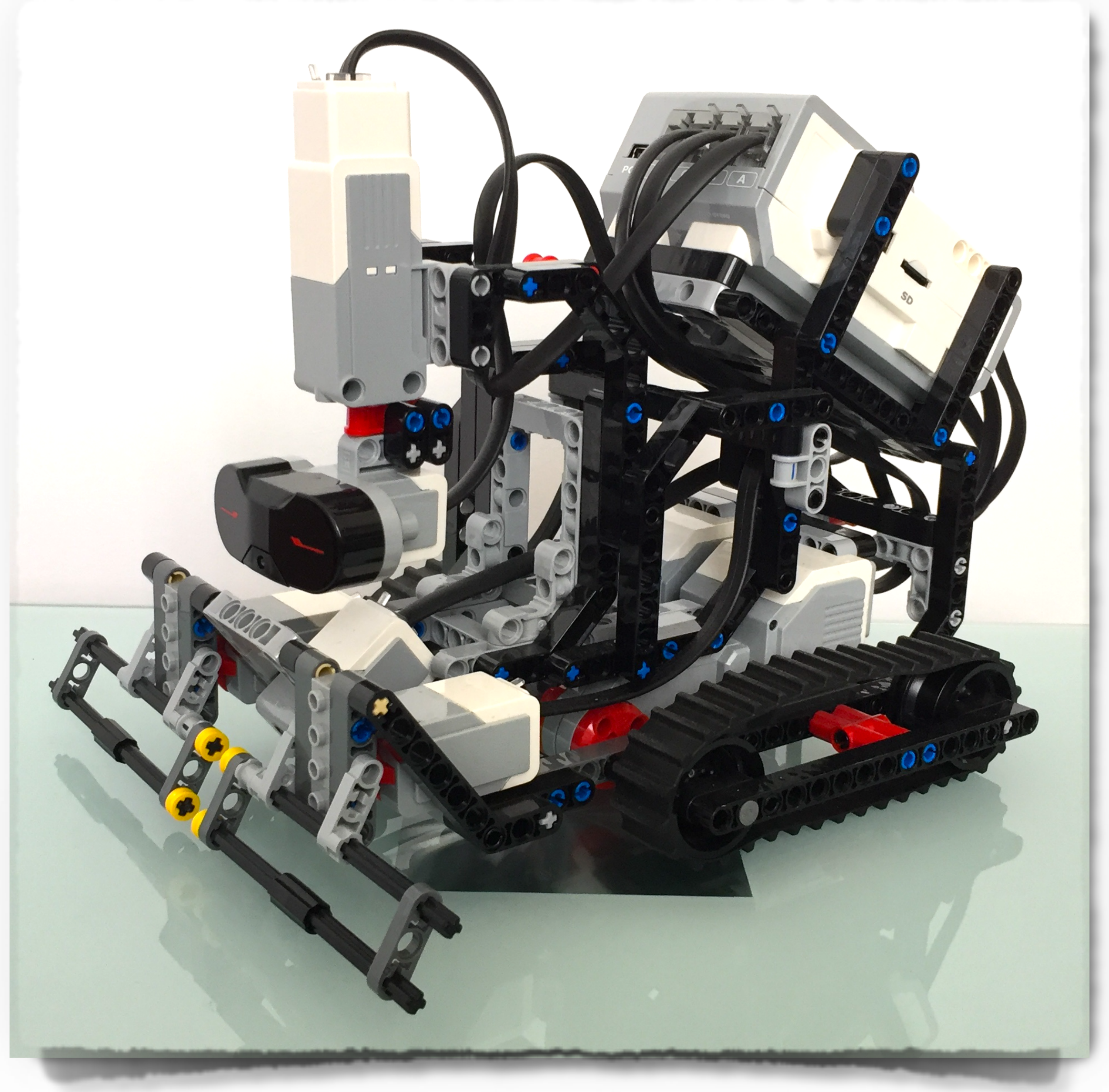


Custom

Courtesy of Tom Lackamp

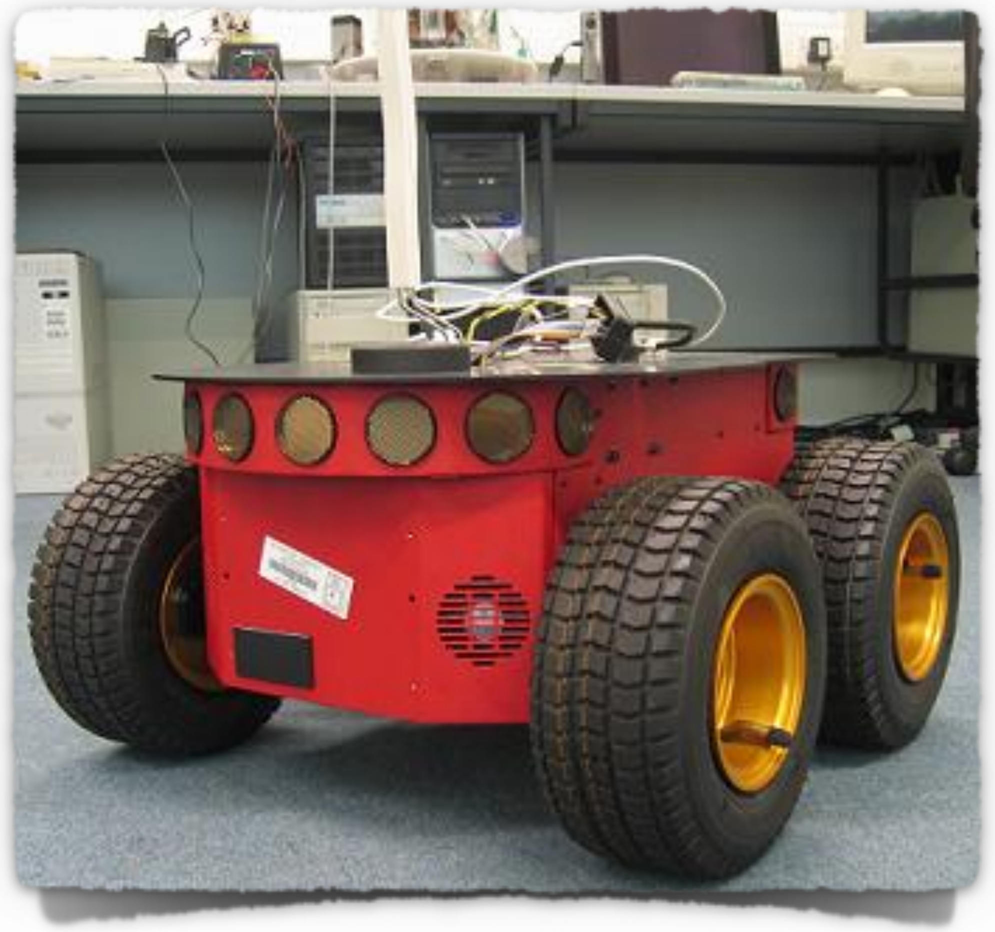
Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Wheeled Robots



- Wheels are a good solution for many applications
  - Three wheels are sufficient to guarantee stability
  - More than three wheels requires flexible suspension

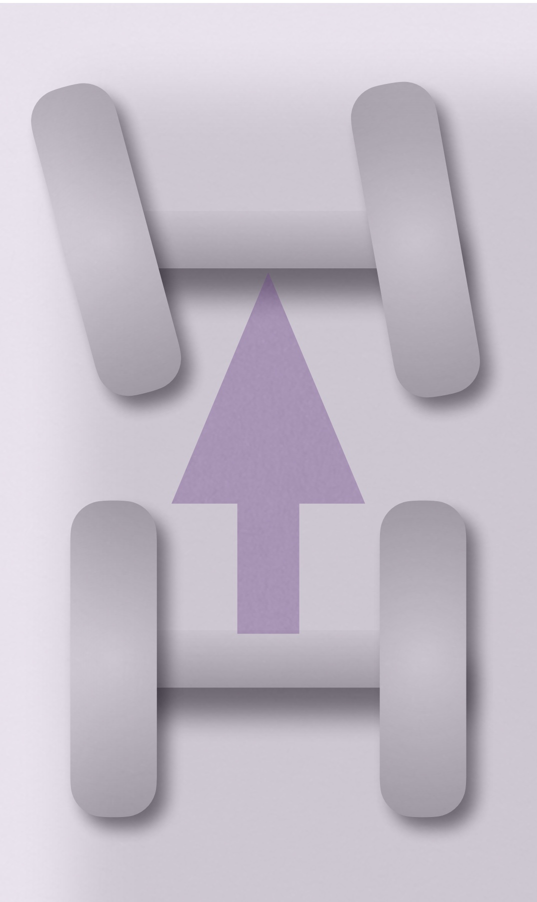
- Different configurations for drive and steering



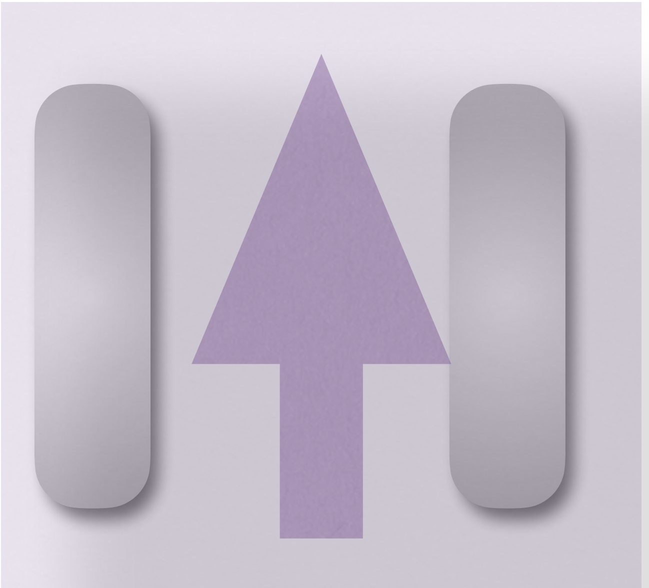
- Tracked robots use slip/skid steering
  - can be controlled with two wheels

Steering and Movement

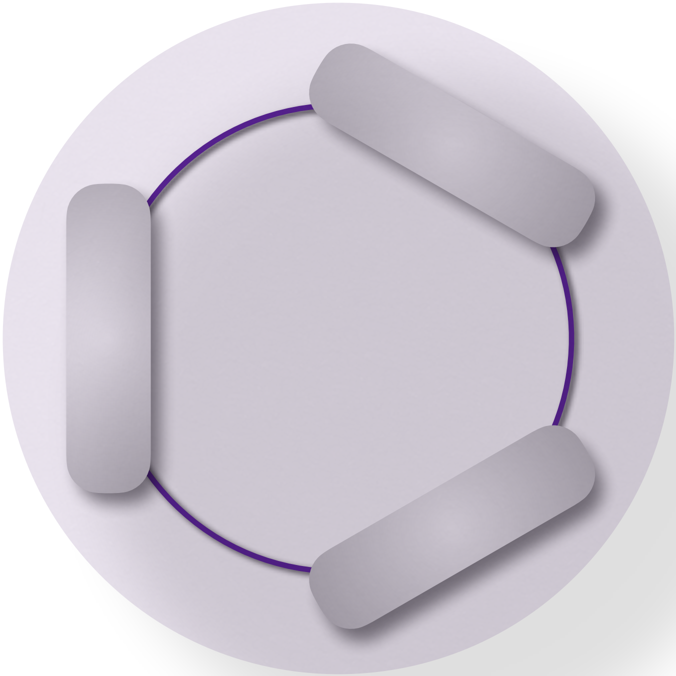
- Three main approaches to steering:
  - Steering wheels at front, with drive wheels at back
    - Similar to a car
  - Differential drive
    - Turning achieved by varying the individual velocity / speed of each wheel
  - Omidirectional drive
    - Can move in any direction, in any orientation
      - Check out this example of an holonomic robot
      - https://youtu.be/-ZdBowwPZas



Steering Wheels



Differential Drive



Omnidirectional Drive

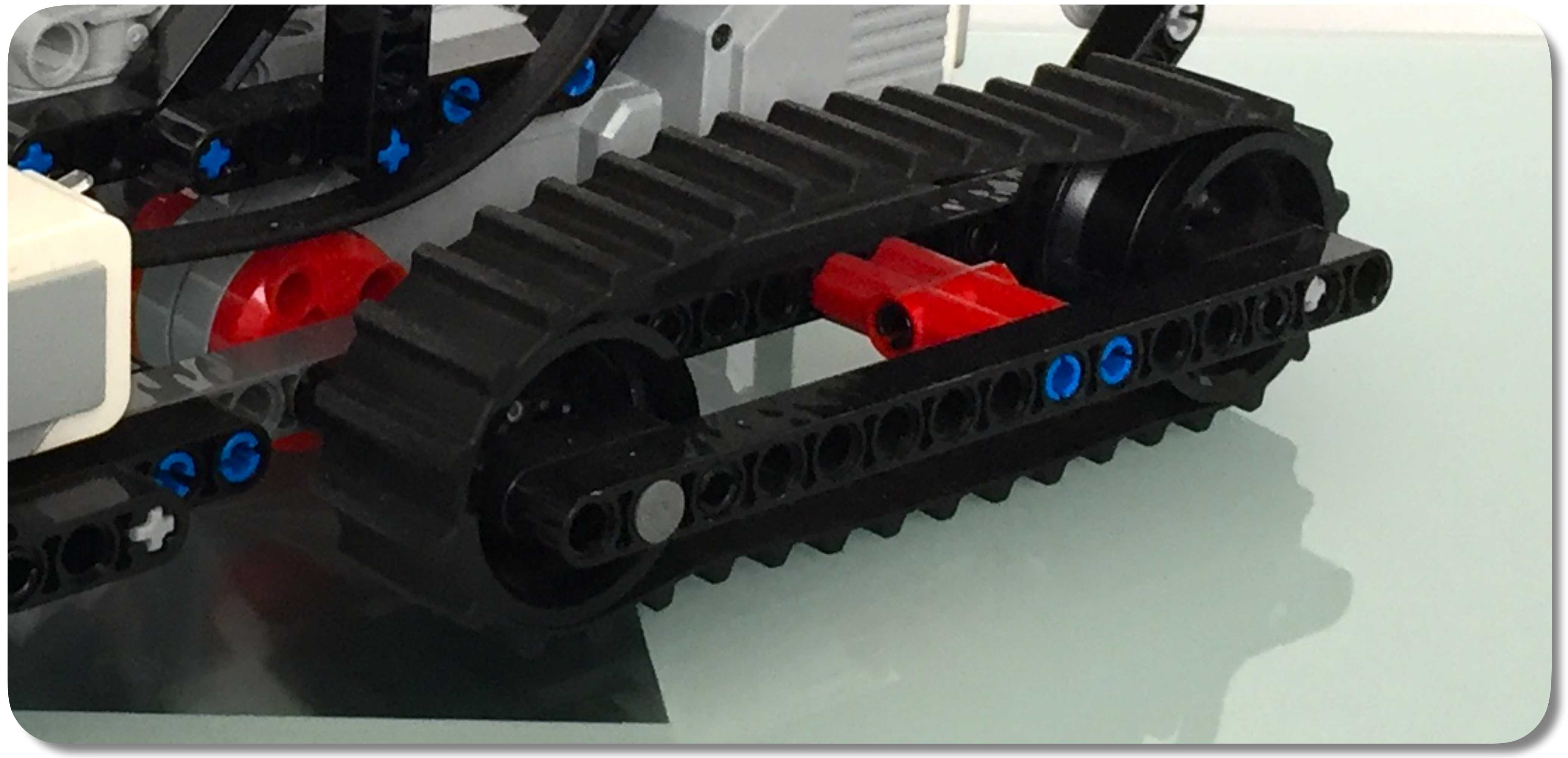
Navigation through Pilots

- Distance information on its own permits a crude form of navigation:
  - *Dead reckoning!!!*
  - Calculate how far the robot has gone based on wheel rotations.
  - Our robot uses a slip/skid drive, which is similar to a differential drive, but with worse odometry.

- LeJOS provides several Pilot classes to support different types of vehicle
  - Three main Pilot classes are currently provided by the `lejos.robotics.navigation` package
    - `MovePilot` - used instead of the depreciated DifferentialPilot class
    - `OmniPilot` - for use with holonomic robots
    - `SteeringPilot` - for use with Steering wheels

Move Pilot

- Constructing a MovePilot
  - Based on the definition of a Chassis
  - Requires the definition of the two wheels, comprising:
    - The wheel diameter
    - Position from the center of the robot (i.e. half of the track width)
      - The track width is the distance between the left and right wheels
    - Motor port
    - Optional gear train between wheel and motor (not used with our robot)
  - Typically requires some trial and error!!!



```
Wheel leftWheel = WheeledChassis.modelWheel(Motor.B, 3.3).offset(-10.0);
Wheel rightWheel = WheeledChassis.modelWheel(Motor.C, 3.3).offset(10.0);
Chassis myChassis = new WheeledChassis(
                new Wheel[]{leftWheel, rightWheel},
                WheeledChassis.TYPE_DIFFERENTIAL);
MovePilot pilot = new MovePilot(myChassis);
```


Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

MovePilot Methods

- Speed of motion (linear or rotation)
  - speed is in wheel-diameters-units per second (e.g. cm per second)
    - `setLinearSpeed(double speed)`
    - `setAngularSpeed(double speed)`
  - Also possible to get current speed
    - e.g. `double getLinearSpeed()`
  - and max possible speed
    - e.g. `double getMaxLinearSpeed()`
  - Also possible to set acceleration, etc

- Move a certain amount
  - `travel(double distance)`
    - distance is in wheel-diameters-units (e.g. cm)

- Rotate:
  - `rotate(double angle)`
    - rotate through specified angle (in degrees) in a zero-radius turn.

- Lots of other methods defined in the API.

Example Code - MovePilot

The `MovePilot` instance is created by generating two instances of the type wheel using the modeller method `modelWheel`. The parameters here broadly represent the robot, but as we have a differential drive, the precise values may need calibrating.

```
public class SimplePilot {
  MovePilot pilot;
  GraphicsLCD lcd;

  public void  drawSquare(float length){
    for(int i = 0; i<4 ; i++){
      pilot.travel(length);      // Drive forward
      pilot.rotate(90);          // Turn 90 degrees
    }
  }

  public static void main(String[] args) {
    Wheel leftWheel = WheeledChassis.modelWheel(Motor.B, 3.3).offset(-10.0);
    Wheel rightWheel = WheeledChassis.modelWheel(Motor.C, 3.3).offset(10.0);
    Chassis myChassis = new WheeledChassis(
          new Wheel[]{leftWheel, rightWheel}, WheeledChassis.TYPE_DIFFERENTIAL);

      // Create a SimplePilot and instantiate its member pilot
      SimplePilot sp = new SimplePilot();
      sp.pilot = new MovePilot(myChassis);
      sp.lcd = LocalEV3.get().getGraphicsLCD();

      sp.pilot.setLinearSpeed(20);      // Set speed to 20cm per second
      sp.drawSquare(40);
  }
}
```

Example Code - MovePilot

The **drawSquare** method draws the four sides of the square, by using **pilot.travel(length)** to move forward the length of a side, and rotating around 90 degrees at each corner by using **pilot.rotate(90)**

The speed of the pilot is defined using the method **setLinearSpeed()** to travel at 20cm per second. A 40cm square is drawn using **drawSquare(40)**

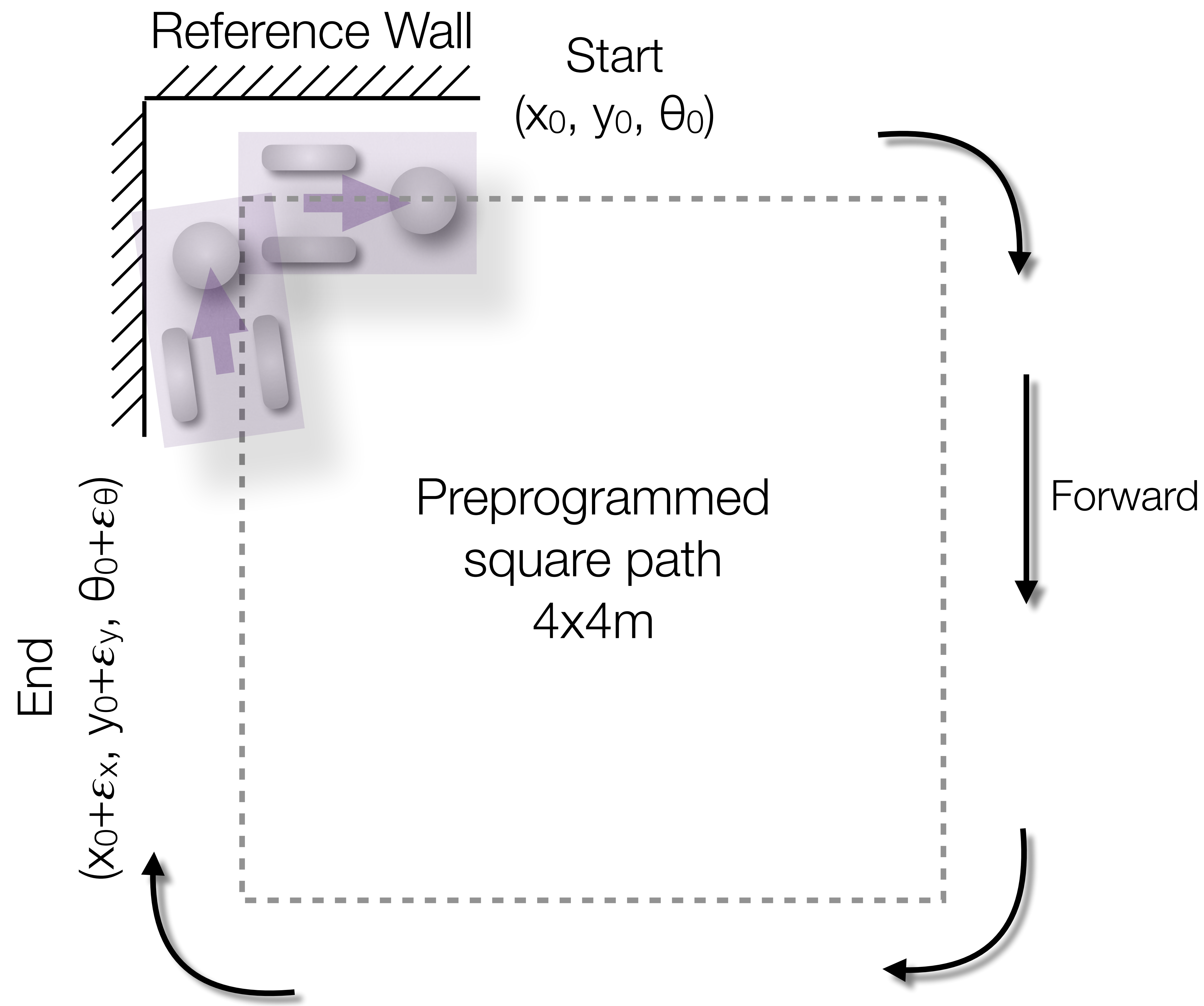
```
public class SimplePilot {
  MovePilot pilot;
  GraphicsLCD lcd;

  public void  drawSquare(float length){
    for(int i = 0; i<4 ; i++){
      pilot.travel(length);      // Drive forward
      pilot.rotate(90);          // Turn 90 degrees
    }
  }

  public static void main(String[] args) {
    Wheel leftWheel = WheeledChassis.modelWheel(Motor.B, 3.3).offset(-10.0);
    Wheel rightWheel = WheeledChassis.modelWheel(Motor.C, 3.3).offset(10.0);
    Chassis myChassis = new WheeledChassis(
          new Wheel[]{leftWheel, rightWheel}, WheeledChassis.TYPE_DIFFERENTIAL);

      // Create a SimplePilot and instantiate its member pilot
      SimplePilot sp = new SimplePilot();
      sp.pilot = new MovePilot(myChassis);
      sp.lcd = LocalEV3.get().getGraphicsLCD();

      sp.pilot.setLinearSpeed(20);      // Set speed to 20cm per second
      sp.drawSquare(40);
  }
}
```

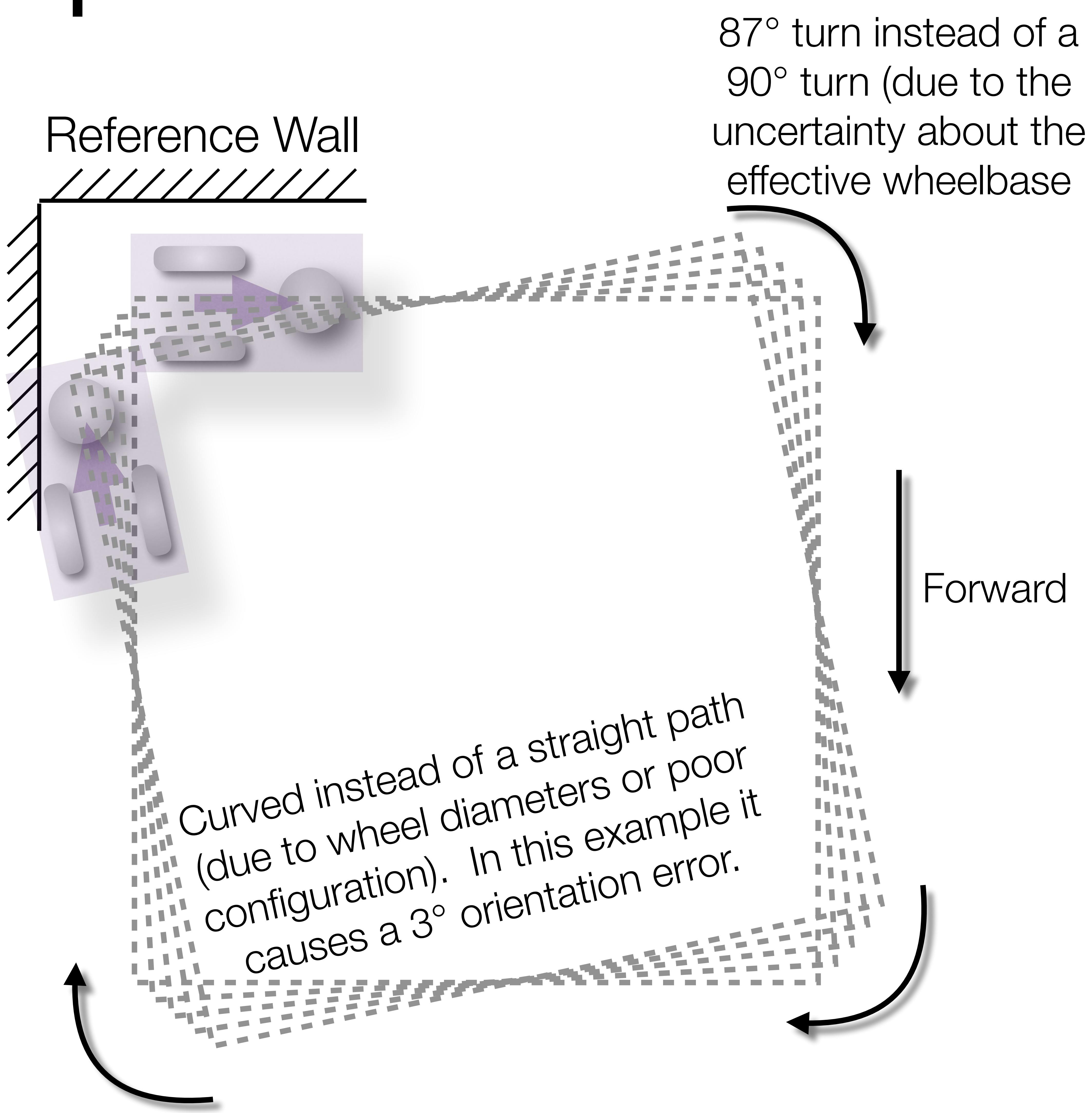
MovePilot - Calibration

- Using the `MovePilot` and the `wheeledChassis`
  - You will find it doesn't do exactly what you ask it to right away.
- With correct robot dimensions
  - Pretty good on distance.
  - Less good on rotation.
- You will need to ***callibrate*** to get it to do what you want it to do.
- You will need to ***keep on*** callibrating.

Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Borenstein's experiment





Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Odometry Pose Provider (OPP)

- When using the `MovePilot`
  - the control loop running the motors knows instantaneously how far the robot has moved.
  - That is what it uses to know when to stop the motors.
- It is useful to be able to log this information in the control program.
  - The `OdometryPoseProvider` provides some of this ability.
- Pose objects are manipulated by `OdometryPoseProvider`
  - Pose objects store a robot pose.
  - Turns out you need to do this a lot. Has no necessary relation to where the robot is.
  - Methods:
    - `getX()`
    - `getY()`
    - `getHeading()`
  - Just as you might/should expect.
  - Values returned are `floats`.

Odometry Pose Provider (OPP)

- Getting and Setting the Pose
  - `void SetPose(Pose aPose)`
    - sets the Pose value in the OPP.
    - Note that this does ***not*** move the robot, just changes the value that is stored.
  - `Pose GetPose()`
    - returns a Pose.
    - This is the current pose stored by the OPP.
    - If used correctly, this `Pose` will tell you something useful.
  - A `Pose` maintains:
    - `float _heading`
    - `Point _location`

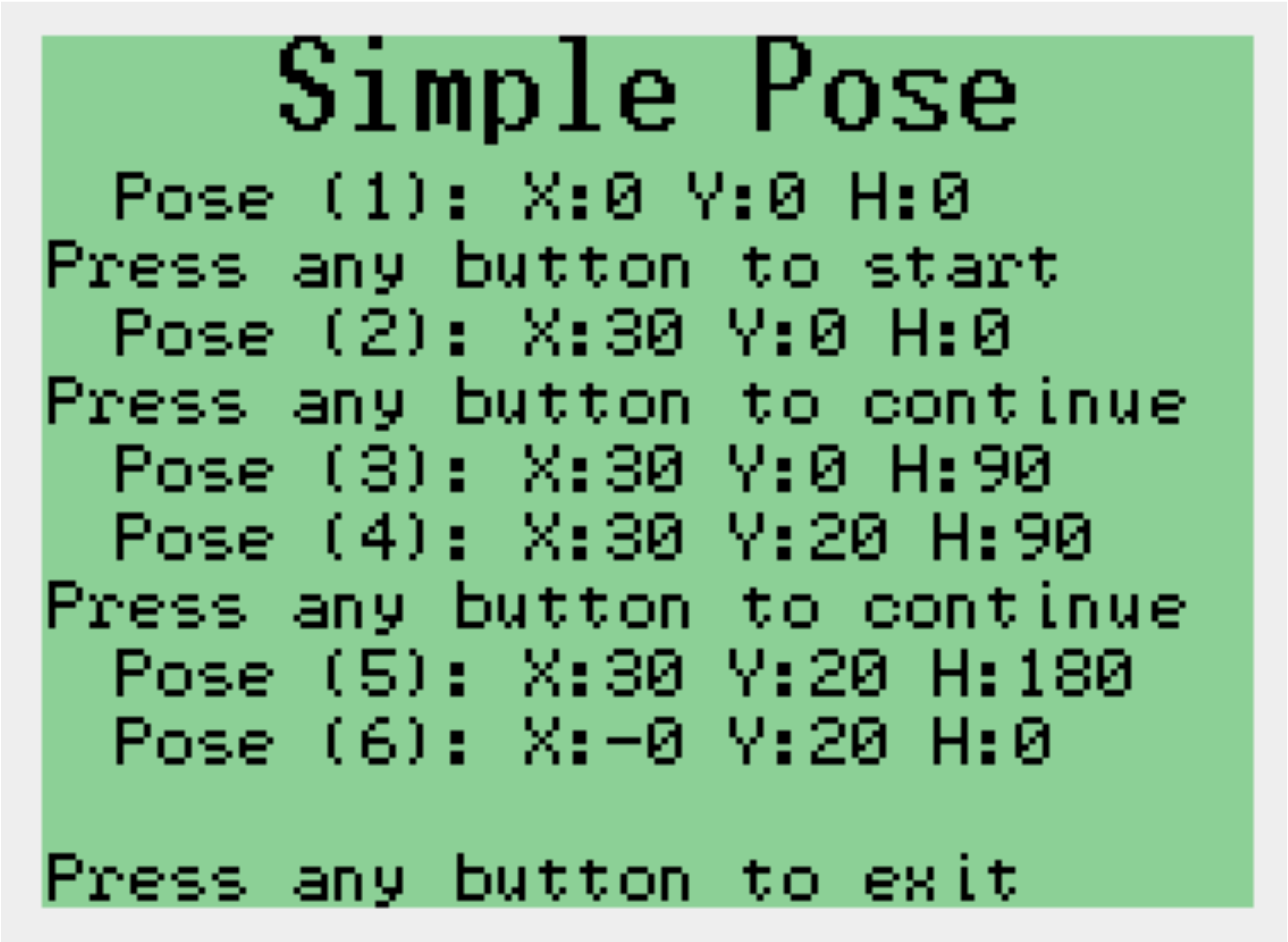
- When you create an OPP, you link it to a Pilot object:

```
OdometryPoseProvider opp =
        new OdometryPoseProvider(pilot);
```

  - where `pilot` is a `MovePilot`.
  - Then the `Pose` returned by the OPP is updated when the robot moves.
  - Of course, it is updated by the amount that the robot thinks it moves.
    - (Since the robot doesn't ***actually know*** how much it is moving.)

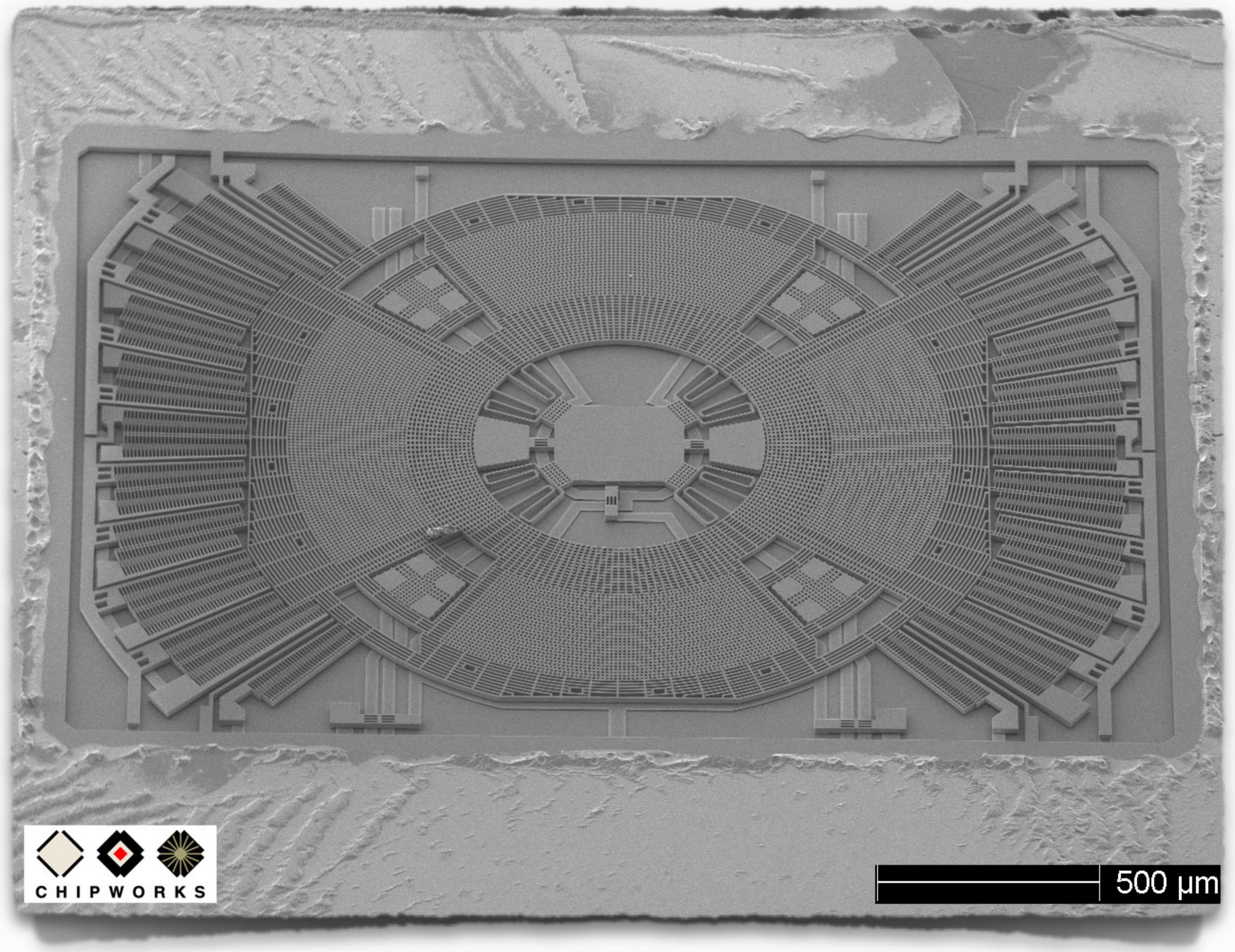
Example Code - OPP

```
// Create a pose provider and link it to the move pilot
OdometryPoseProvider opp = new OdometryPoseProvider(pilot);

lcd.drawString("Pose (1): " + opp.getPose(), 10, 20, 0);
pilot.travel(30);
lcd.drawString("Pose (2): " + opp.getPose(), 10, 40, 0);
pilot.rotate(90);
lcd.drawString("Pose (3): " + opp.getPose(), 10, 60, 0);
pilot.travel(20);
lcd.drawString("Pose (4): " + opp.getPose(), 10, 70, 0);
pilot.rotate(90);
lcd.drawString("Pose (5): " + opp.getPose(), 10, 90, 0);
pilot.travel(30);
pilot.rotate(-180);
lcd.drawString("Pose (6): " + opp.getPose(), 10, 100, 0);
```



Heading Sensors

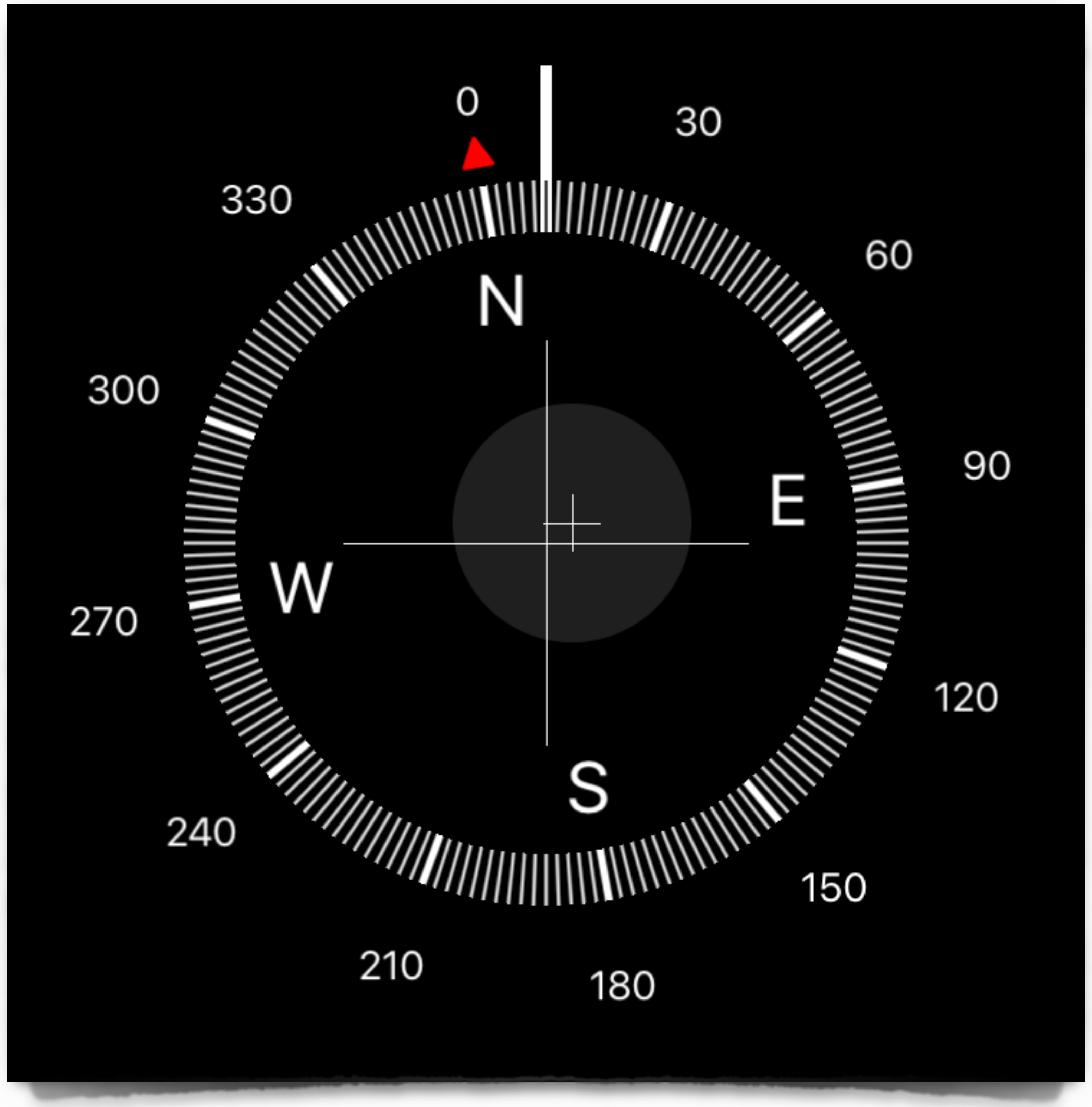
- Heading sensors can be:
  - proprioceptive (gyroscope, inclinometer); or
  - exteroceptive (compass).
- Used to determine the robot's orientation and/or inclination.
  - Allow, together with an appropriate velocity information, to integrate the movement to an position estimate.
- A bit more sophisticated than just using odometry.



Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Compass

- Used since before 2000 B.C.
  - Chinese suspended a piece of naturally magnetic magnetite from a silk thread and used it to guide a chariot over land.
- Magnetic field on earth
  - Absolute measure for orientation.
- Large variety of solutions to measure the earth's magnetic field
  - Mechanical magnetic compass
  - Direct measure of the magnetic field
    - (Hall-effect, magnetoresistive sensors)



Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Compass

- Major drawback
  - Weakness of the earth field
  - Easily disturbed by magnetic objects or other sources
  - Not feasible for indoor environments in general.

- Modern devices can give 3D orientation relative to Earth's magnetic field.

Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

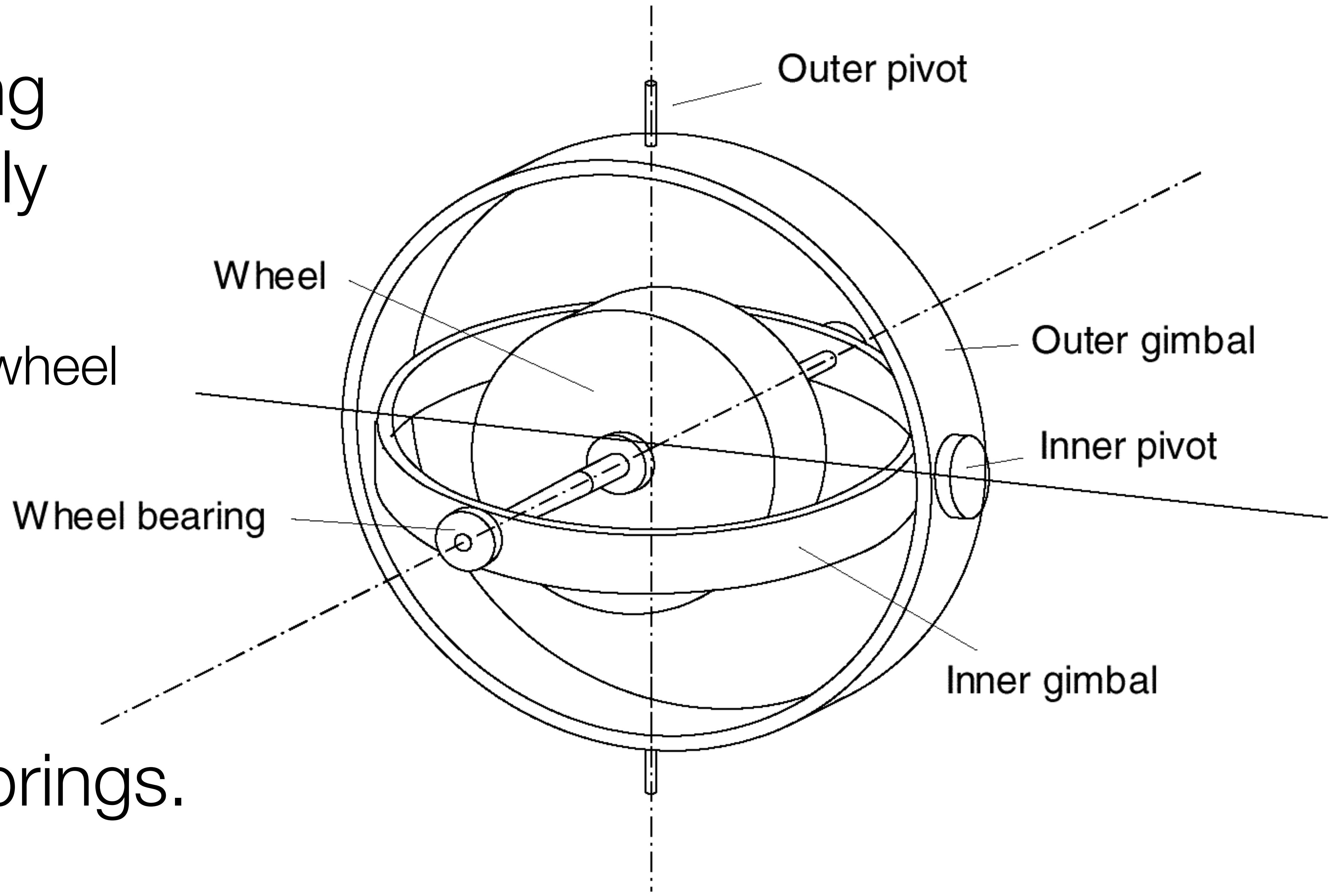
Gyroscope

- Heading sensors, that keep the orientation to a fixed frame
  - Provide an absolute measure for the heading of a mobile system.
  - Unlike a compass doesn't measure the outside world.
- Two categories, mechanical and optical gyroscopes
  - ***Mechanical*** Gyroscopes
    - Standard gyro
    - Rate gyro
  - ***Optical*** Gyroscopes
    - Rate gyro

Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Mechanical Gyroscopes

- Concept: inertial properties of a fast spinning rotor
  - gyroscopic precession
- Angular momentum associated with a spinning wheel keeps the axis of the gyroscope inertially stable.
  - No torque can be transmitted from the outer pivot to the wheel axis
  - Spinning axis will therefore be space-stable
  - Quality: 0.1 degrees in 6 hours
- In ***rate*** gyros, gimbals are held by torsional springs.
  - Measuring force gives angular velocity.

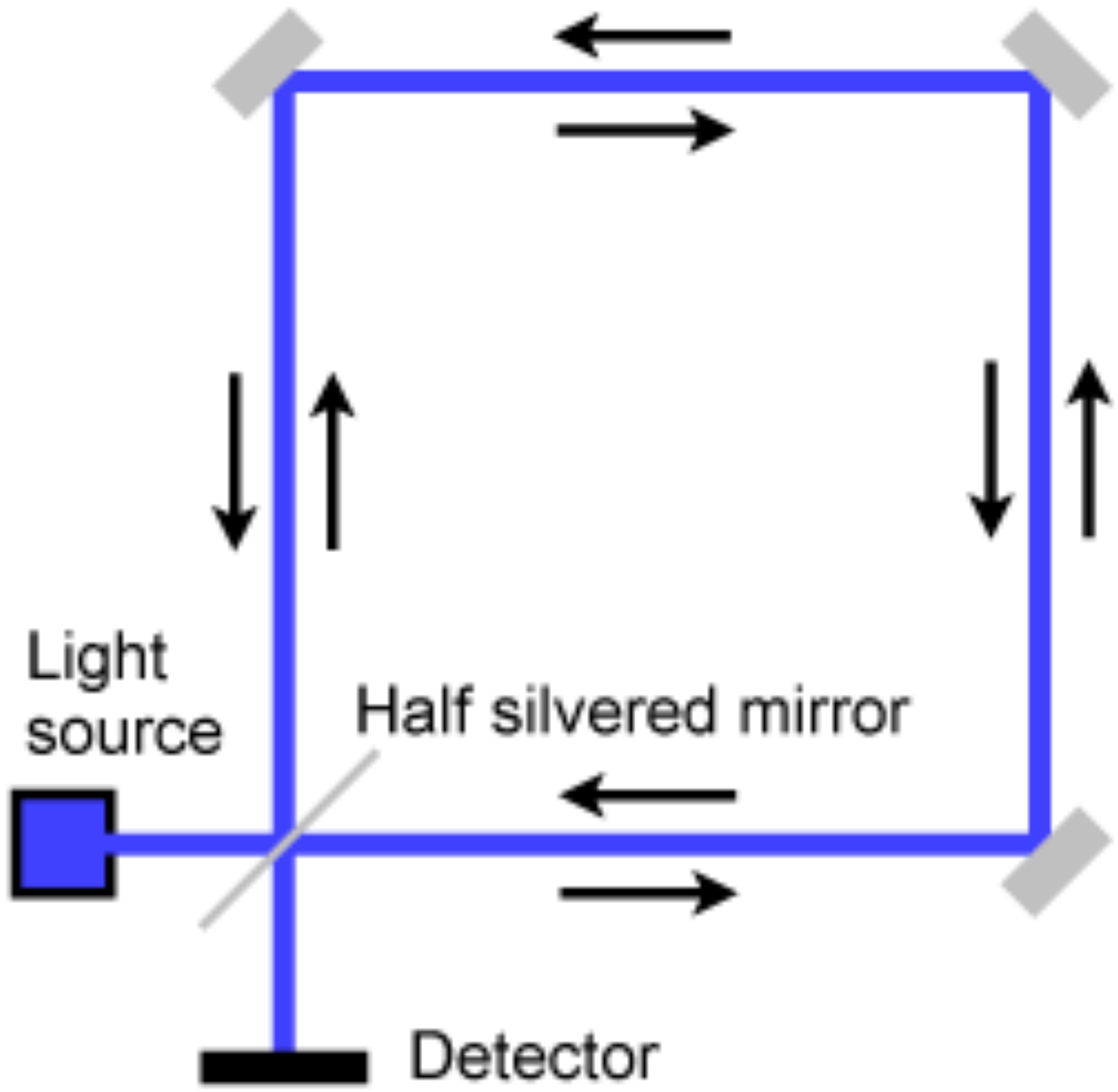


Optical Gyroscopes

- Use two monochromatic light (or laser) beams from the same source.

- One beam travels clockwise in a cylinder around a fibre, the other→ counterclockwise.
  - The beam traveling in direction of rotation:
  - Slightly shorter path shows a higher frequency
  - Difference in frequency Δf of the two beams is proportional to the angular velocity Ω of the cylinder/fibre.

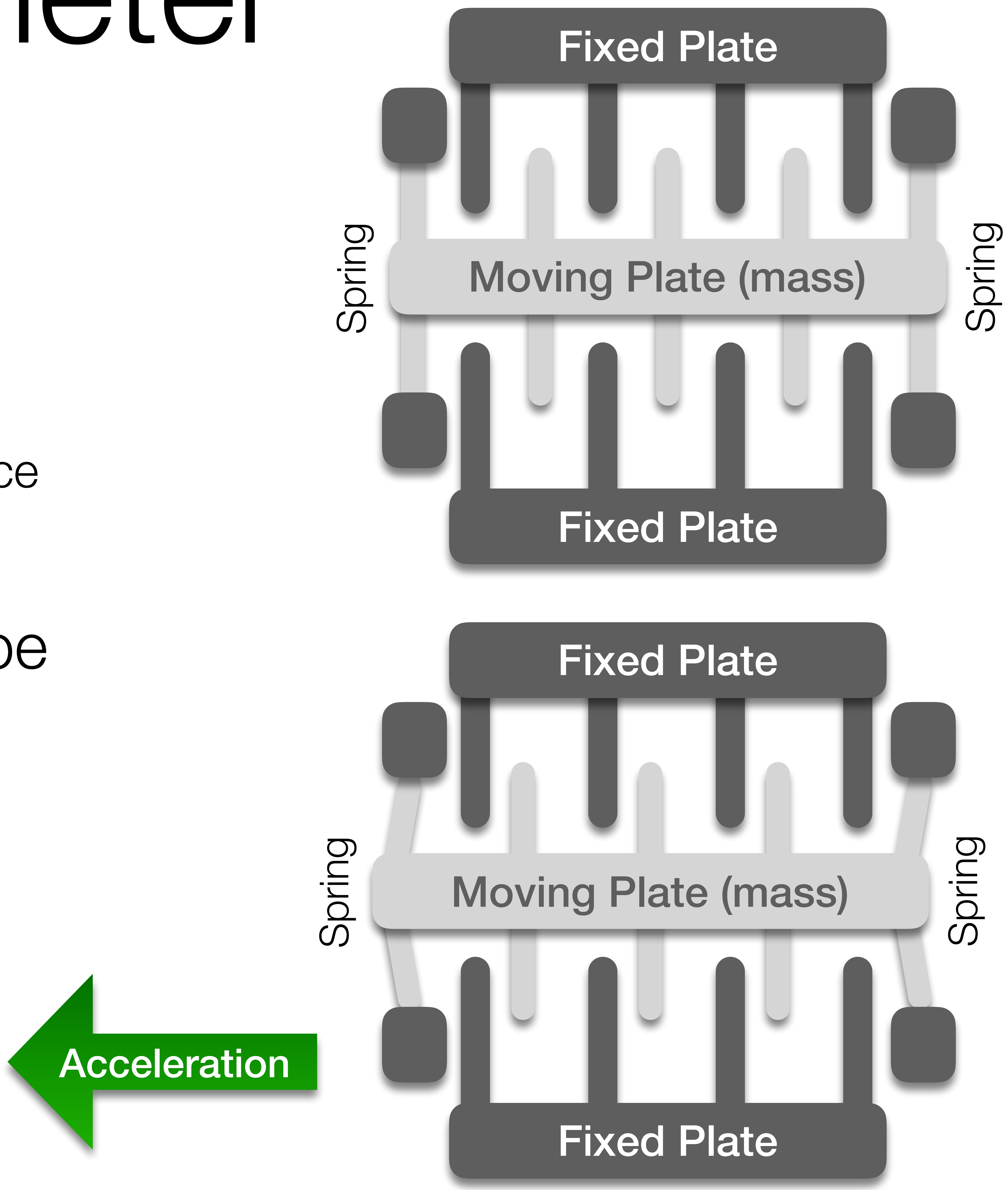
- Newest optical gyros are solid state.



Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Accelerometer

- Measure acceleration.
  - Mass on a spring.
    - Measure force in spring through the change in capacitance
    - Gives acceleration of mass.
  - Any heading or position sensor reading can be "differentiated" to give acceleration.
    - Difference between two values gives "velocity"
    - Difference between two "velocities" gives acceleration
  - Any velocity sensor reading can be handled similarly.



Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Sensor Performance

- Dynamic Range
  - Spread between lower and upper limits of input values (as a ratio)
- Resolution
  - Minimum difference between two sensor values
- Sensitivity
  - Measure of the degree to which incremental change in target input changes output signal
    - Ratio of output change to input change
  - In real world environment, the sensor has very often high sensitivity to other environmental changes, e.g. illumination.
- Cross-sensitivity
  - Sensitivity to environmental parameters that are orthogonal to the target parameters
- Error / Accuracy
  - Difference between the sensor's output and the true value

$$error = m - v$$

$$accuracy = 1 - \frac{|m - v|}{v}$$

  - where
    - m = measured value and
    - v = true value.

Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Sensor Performance

- Systematic error → deterministic errors
  - Caused by factors that can (in theory) be modelled
    - → prediction
  - e.g. distortion caused by the optics of a camera.
- Random error → non-deterministic
  - No prediction possible
  - However, they can be described probabilistically
    - e.g. error in wheel odometry.
- Precision
  - Reproducibility of sensor results
  - If a random error of a sensor is characterised by some mean σ and standard deviation, μ then the precision is the ratio of the sensors output range to s.d.

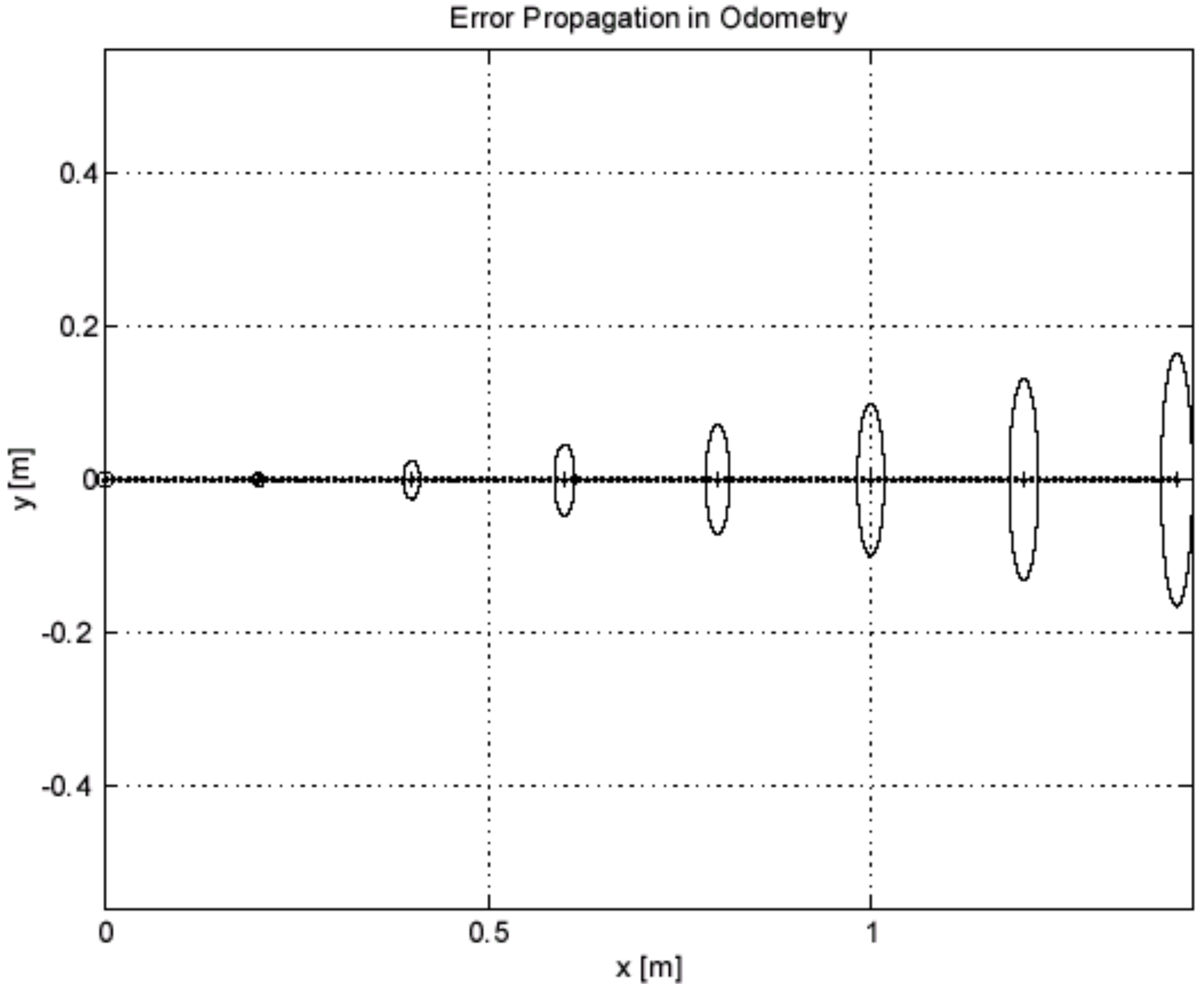
$$precision = \frac{range}{\sigma}$$

Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Coping with Errors

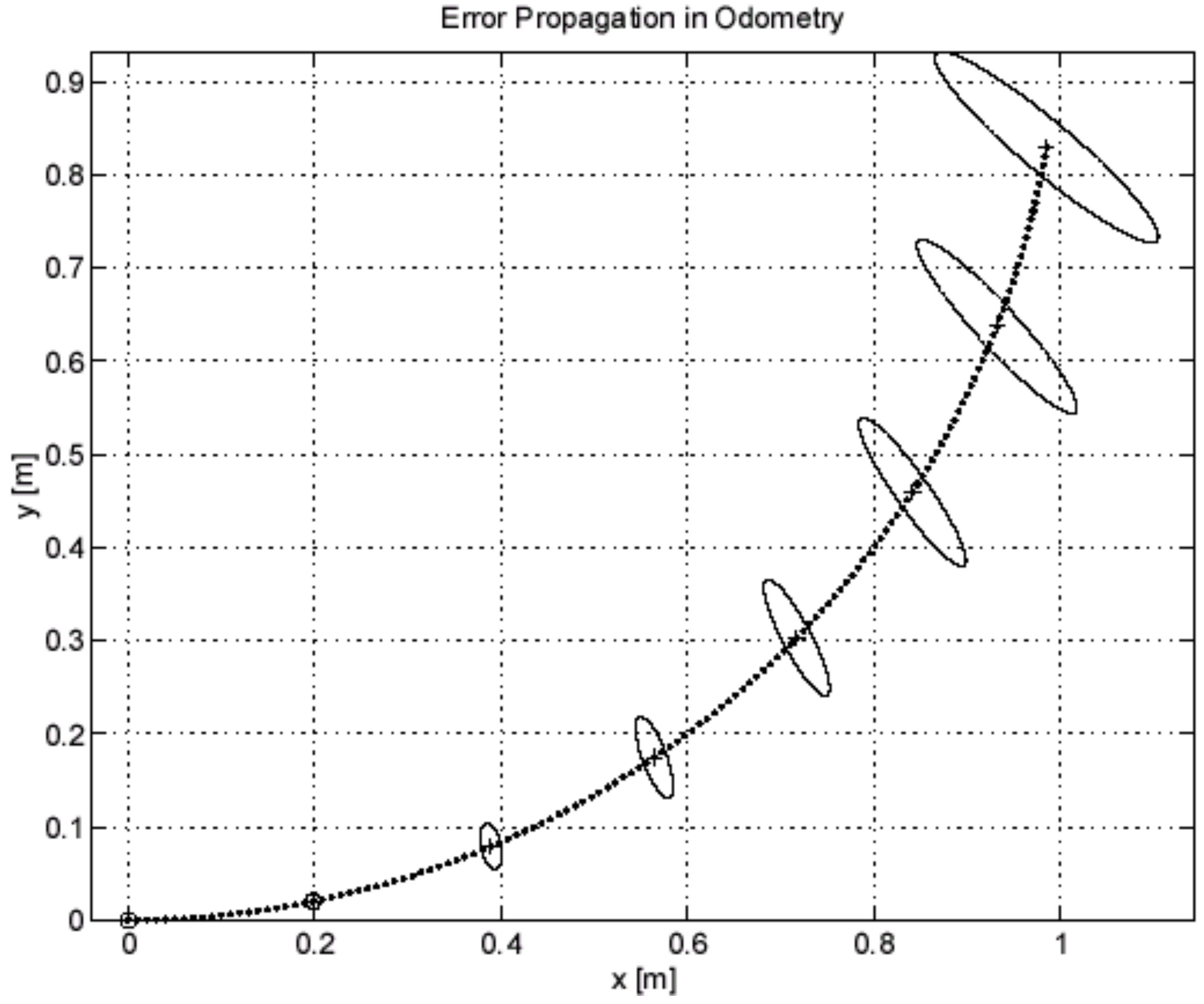
- Compensate for systematic errors.
  - Build a probabilistic model of random errors.
  - Obtain a distribution of possible positions.

- Know where we are ***on average***.
  - Don't know where we are ***in particular***.
  - Errors accumulate over time.



A simple error model for straight line motion

(Thrun, Burgard and Fox)



A simple error model when we turn

(Thrun, Burgard and Fox)

Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17

Summary

- This lecture started to look at sensor data.
  - It concentrated on data that can be used in odometry.
    - Wheel encoders
  - and looked at LeJOS support for doing odometry.
  - Also looked at other kinds of related sensor data:
    - Compass
    - Gyroscope
  - Later in the module we will look at range sensor data and cameras as sensors.

- In the next lecture, we will look at Behaviour Based Robots

Original Source: M. Wooldridge, S.Parsons, D.Grossi - updated by Terry Payne, Autumn 2016/17